Commentaires DM n°4 : Gestion d'un allocateur dynamique de mémoire en C

Généralités

- problème qui commence à ressembler à un sujet de concours ; ne pas attendre le dernier moment pour y réfléchir ; et en profiter pour affiner la réflexion et soigner la rédaction
- problème progressif sur le fonctionnement d'un "tas" (allocation dynamique de mémoire)
- attention à la lecture de l'énoncé : les consignes sur l'utilisation des fonctions étaient précises, et les exemples devaient être parfaitement compris
- quelques copies trop brouillonnes, trop de copies avec du code sans aucune explication

Introduction

- ne pas confondre l'utilisateur du "service" (le programmeur qui va demander et libérer de l'espace mémoire) et l'implémentation du "service" (4 implémentations, de plus en plus complexes)
- `p` désigne toujours l'indice du début de zone utile ; selon les implémentations, il y a des informations avant et ou après la zone utile

Implémentation naïve

A priori facile, pour se familiariser avec le problème : on écrit les portions mémoire les unes à la suite des autres

- **Q1**. on peut écrire directement dans le tableau `mem[p+i] = c` ; `ecrire(p,i,c)` concerne l'utilisateur du service
- **Q3**. complexité en fonction de plusieurs paramètres. Il y a 2 cas : espace mémoire insuffisant, espace mémoire suffisant. Attention, on raisonne à n fixé (le pire des cas n'est pas "lorsque $n = TAILLE\_MEM$".

Avec des blocs de taille fixe

On écrit les portions mémoire dans des blocs de taille fixe, en recyclant si possible.

On doit utiliser les fonctions proposées `lire_prochain`, `ecrire_prochain`, ...

- **Q5**. algorithme de réservation : si la taille demandée est trop grande, -1 ; sinon, on parcourt tous les blocs à la recherche d'un bloc libre. S'il n'y a pas de bloc libre, on rajoute un bloc (risque de mémoire pleine)

Portions avec en-tête et pied de page

Ca devient compliqué ... en-tête et pied de page sont égaux ; le pied de page permet de passer simplement à la portion précédente

Codage : les tailles sont forcément paires et le bit de poids faible permet d'indiquer que la portion est libre ou réservée

Prologue-épilogue : permettent d'éviter de sortir de la mémoire dans les algorithmes de réservation et libération

- **Q7**. explications : essayer d'être clair et concis ...
- **Q8**. démarrage : on réserve le prologue, l'épilogue et on indique la position de l'épilogue
- **Q9**. algorithme de réservation : on parcourt les portions à la recherche d'une portion libre suffisamment grande ; si on en trouve une, on la coupe éventuellement en 2 ; si on n'en trouve pas, on déplace l'épilogue (s'il reste suffisamment de place) ;
- **Q9**. petits détails : la taille réservée n doit être ajustée à un entier pair ; mais on initialise sur n emplacements
- **Q10**. algorithme de libération : invariant : il n'y a jamais 2 portions libres côte à côte. Lorsqu'on libère une portion, on regarde les portions voisines pour éventuellement les fusionner. Au final il n'y a qu'un seul appel à `marque_libre` et l'algorithme est en $\mathcal{O}(1)$

Avec chaînage explicite des portions libres

Dans la partie précédente, la recherche d'une portion libre peut être coûteuse. On imite une liste doublement chaînée pour gagner un peu de temps.

L'information est stockée dans les deux premières cases de chaque portion libre (les portions manipulées étant de taille au moins 2).

Pour les listes (doublement) chaînées, un schéma est utile.

- **Q11**. ajout en tête : on modifie la tête de chaîne ; ne pas oublier de modifier le prédécesseur de l'ancienne tête de chaîne ; attention au cas particulier de la liste vide
- **Q12**. suppression : on modifie le successeur du prédécesseur ; on modifie le prédécesseur du successeur ; attention aux cas particuliers : suppression du premier ou dernier élément
- **Q13 - Q14 - Q15**. c'est quasiment le même code que dans partie précédente ; à la réservation, on parcourt uniquement la liste des portions libres ; on fait attention à supprimer de la liste (lorsqu'on recycle), et à ajouter dans la liste (lorsqu'on libère). La complexité dans le pire des cas reste la même, mais on peut s'attendre à ce qu'en pratique ce soit plus efficace.